DISASM USER'S MANUAL

VERSION 2.0

AN INTELLIGENT DISASSEMBLER FOR THE APPLE II

SECTION	TITLE
1	INTRODUCTION
2	USING DISASM
3	HOW IT WORKS
4	EXAMPLE DISASSEMBLY
5	MEMORY USEAGE
6	ADD'L INFO

RAK-WARE

41 RALPH ROAD

WEST ORANGE NJ 07052

© MAR, 1981

THE ORIGINAL AND ANY BACKUP COPIES OF RAK-WARE PROGRAMS ARE INTENDED FOR YOUR PERSONAL USE IN CONNECTION WITH A SINGLE COMPUTER. YOU MAY NOT DISTRIBUTE COPIES OF OR ANY PART OF EITHER THIS DOCUMENTATION OR THE SOFTWARE TO OTHERS WITHOUT SPECIFIC APPROVAL FROM RAK-WARE.

*** DISASM (2.0) User's Manual ***

SECTION 1 - INTRODUCTION

DISASM is a utility program for the Apple II which creates a source module from machine code (also known as object code). It is intended as an aid for understanding and modifying machine language programs.

DISASM is an intelligent 2-pass disassembler-with-labels which provides the user with a text file which is compatible with, and easily read by several of the most popular assemblers currently available for the Apple II.

DISASM has the following performance features:

- 100% machine language for fast operation (no Integer Basic or Applesoft drivers required)
- Operates either on an Apple II or Apple II Plus
- Selectable output formats are directly compatable with DOS Toolkit, LISA and S-C assemblers
- Correctly disassembles displaced object code
- Labels are assigned in accordance with address catagorization (i.e. Page Zero, External or Internal)
- Labels and addresses sorted for user convenience
- Optional User-defined label name table provides more meaningful label assignments
- Equate definitions generated for Page Zero and External address references
- Auto source segmentation for easier reading & understanding
- Undefined opcodes replaced with Hex byte definitions
- Captures software tricks for hiding code
- No restriction on disassembled block length
- Error checking provided for both user response and internal operations
- Easy to use! Prompting minimizes need for consulting user's manual

SECTION 2 - USING DISASM

DISASM is easy to use as outlined in the following steps:

1. Turn-on Apple and boot-up with the DISASM program diskette. (The assembler need not be resident to use DISASM.) NOTE: The program diskette is in 16-sector format. (13-sector diskettes are also available on request.)

2. The 'HELLO' program will remind you to BLOAD both DISASM and the Monitor ROM label name table. The user can now load the program to be disassembled into RAM. (See MEMORY USEAGE to avoid conflicts.)

3. Run DISASM either from Basic or the Monitor with 'CALL 2048' or '800G' respectively.

4. DISASM will prompt you to choose an output format. Each selection listed has somewhat different characteristics which are peculiar to that assembler. Select the one you need (no RETURN is necessary).

5. Next you will be asked to provide certain information about the object code to be disassembled. First type in the beginning and the ending addresses of where the object code is physically located in memory. (Each address must be in hexidecimal followed by a RETURN. Non-hex entries or entries greater than 4 characters are rejected.) Next, enter the address at which the object code actually executes. If the physical and execution addresses are the same then just type RETURN.

6. Next enter the starting location of the User-defined label name table. Type in '2000' if you wish to use the Monitor ROM table provided on the DISASM program diskette (see ADD'L INFO for more details on the label name table). Use of the label table is optional. Just type RETURN if you don't want to use this feature.

7. Finally, you must enter the name to be assigned to the text file generated by DISASM. The name should be limited to a reasonable length since only the first 40 typed characters are recognized. Qualifiers such as ',S6,D2,V100' may be appended to the file name, if needed. (Make sure there is a diskette in the disk drive before hitting RETURN.)

SECTION 2 (Con't)

The rest is automatic. DOS commands and disassembled source code are displayed to indicate the sequence of DISASM operations.

The first pass builds and sorts the address tables. Pass 1 execution time is highly dependent on the length of the block being disassembled. This operation may take anywhere from a few seconds (for 1 to 2K blocks) up to several minutes (for 8K and larger blocks).

The second pass writes the text file to disk. The time to write the file is primarily dependent on the speed of DOS and on the number of characters in the text file. This could also take several minutes for very long blocks.

After the text file has been written, load and run your assembler. The text file generated by DISASM is read into the assembler simply by typing the appropriate command:

ASSEMBLER	COMMAND TO READ TEXT FILE
DOS TOOLKIT	LOAD xxxx
S-C ASSEMBLER	EXEC xxxx
LISA	D^C EXEC xxxx

where: xxxx = text file name

D^C = control D

Once you've loaded the text file into the assembler, you should check the following:

a) Insert ORG (and OBJ) directives at the beginning of the file.

b) Append an END directive, if required by your assembler.

c) Delete the 'empty' line at the end of the file (may or maynot exist depending on the assembler).

d) Convertany of Apple's standard mnemonics to the syntax required by your assembler if differences exist.

e) Add labels missed due to 'hidden' code (see Section 6).

SECTION 3 - HOW IT WORKS

DISASM scans through the memory range specified by the user twice. The first pass is used to generate address tables which contain all of the referenced addresses within the designated object code block. These addresses are determined via Apple's INSDSP routine which is used to disassemble each instruction. Page zero references or relative branches are related to 2-byte instructions where the address is defined by the second byte. Absolute references occur as 3-byte instructions where the address is defined by the second and third bytes. Three seperate address tables are maintained for page zero, internal and external address These tables help to organize address types for name assignments and are also used to create the 'equate' definitions. (An internal address is defined as one which is contained within the specified execution range.)

A second pass is used to generate the actual text file line by line. Once again the INSDSP routine is used to disassemble each instruction. Characters normally output to the screen are loaded into a buffer. First the value of the program counter is compared to the addresses in the internal reference table. If a match is found then a leading label is output. Next the mnemonic is retrieved from the buffer and output along with any required tab characters. Finally the operand is inspected and if found to be a memory reference (i.e. not immediate or implied accumulator) then the three tables are searched and the appropriate label is substituted. The labels consist of a leading letter prefix (Z,X or I) to indicate address type, a '.' and then either two or four characters. These characters represent the hex value of the addresses they refer to. If the User-defined label name table feature is in effect (Step 6 above) then the names in that table will be substituted for the standard DISASM label assignments.

SECTION 4 - EXAMPLE DISASSMEBLY

The following arbitrary example illustrates the use and features of DISASM.

First, this is how the code would appear if the Apple's disassembler were used:

```
4800-   A5 02        LDA  $02
4802-   F0 0E        BEQ  $4812
4804-   A0 00        LDY  #$00
4806-   20 1B 48     JSR  $481B
4809-   A5 03        LDA  $03
480B-   20 DA FD     JSR  $FDDA
480E-   20 8E FD     JSR  $FD8E
4811-   60           RTS
4812-   A5 4C        LDA  $4C
4814-   85 00        STA  $00
4816-   A5 4D        LDA  $4D
4818-   85 01        STA  $01
481A-   60           RTS
481B-   B9 2A 48     LDA  $482A,Y
481E-   F0 06        BEQ  $4826
4820-   20 ED FD     JSR  $FDED
4823-   C8           INY
4824-   D0 F5        BNE  $481B
4826-   20 8E FD     JSR  $FD8E
4829-   60           RTS
482A-   D7           ???
482B-   B2           ???
482C-   D7           ???
482D-   B3           ???
482E-   00           BRK
```

Using DISASM, we would respond to the prompts as shown in Figures 1,2 and 3 (assuming that the text file was to be used with the S-C Assembler). Next the assembler is loaded and run. The text file, DEMO, is loaded into the S-C Assembler by typing EXEC DEMO. By simply adding an origin directive, the assembled source looks like this:

SECTION 4 (Con't)

```
                        0990          .OR $4800
0000-                   1000 Z.00     .EQ $00
0001-                   1010 Z.01     .EQ $01
0002-                   1020 Z.02     .EQ $02
0003-                   1030 Z.03     .EQ $03
004C-                   1040 Z.4C     .EQ $4C
004D-                   1050 Z.4D     .EQ $4D
                        1060 *
FD8E-                   1070 CROUT    .EQ $FD8E
FDDA-                   1080 PRBYTE   .EQ $FDDA
FDED-                   1090 COUT     .EQ $FDED
                        1100 *
4800- A5 02             1110          LDA Z.02
4802- F0 0E             1120          BEQ I.4812
4804- A0 00             1130          LDY #$00
4806- 20 1B 48          1140          JSR I.481B
4809- A5 03             1150          LDA Z.03
480B- 20 DA FD          1160          JSR PRBYTE
480E- 20 8E FD          1170          JSR CROUT
4811- 60                1180          RTS
                        1190 *
4812- A5 4C             1200 I.4812   LDA Z.4C
4814- 85 00             1210          STA Z.00
4816- A5 4D             1220          LDA Z.4D
4818- 85 01             1230          STA Z.01
481A- 60                1240          RTS
                        1250 *
481B- B9 2A 48          1260 I.481B   LDA I.482A,Y
481E- F0 06             1270          BEQ I.4826
4820- 20 ED FD          1280          JSR COUT
4823- C8                1290          INY
4824- D0 F5             1300          BNE I.481B
4826- 20 8E FD          1310 I.4826   JSR CROUT
4829- 60                1320          RTS
                        1330 *
482A- D7                1340 I.482A   .HS D7
482B- B2                1350          .HS B2
482C- D7                1360          .HS D7
482D- B3                1370          .HS B3
482E- 00                1380          BRK
```

The assembler's source code editing commands can be used to help reassign selective labels to more meaningful names, if desired.

SECTION 5 - MEMORY USEAGE

Memory useage falls into one of four catagories: Instructions, Address Tables, Buffers and Page Zero. The following table lists the memory regions used by DISASM.

INSTRUCTIONS	$ 8ØØ - 1ØFF
ADDRESS TABLES	$11ØØ - ??
BUFFERS	$ 2ØØ - 2FF
PAGE ZERO	$ ØØ - 1B
NAME LABEL TABLE	USER-DEFINED ($2ØØØ-23FF on DISASM pgm disket

Since there is no intrinsic limitation as to the size of the address label tables, the user must exercise caution when disassembling long blocks. DISASM does not check for memory conflicts. A reasonable rule-of-thumb would require one page of address tables for every four pages (1K) of disassembled object code. Thus you should leave sufficient room above the address tables to avoid clobbering either your label name table or your object code.

DISASM gives you some indication as to the memory for the address tables during the PASS 1 operation. As the tables are being formed, the page numbers are output to the screen. (This also lets you know that DISASM is actually working when disassembling those long blocks.)

SECTION 6 - ADD'L INFO

Finding 'Hidden' Code: Occasionally, a programmer uses certain tricks to facilitate efficient code. One of these tricks deals with a method of controlling the initialization of a register. Let's say we wanted to do a LDX #00 for a default entry into a routine but would like a LDX #80 under certain conditions. This could be written as:

```
                    As Assembled          As Disassembled

DEFAULT             LDX #00               LDX #00
                    HEX 2C                BIT $8ØA2
SPECIAL             LDX #80
```

The 2C turns the second LDX into a kindof do-nothing operation and a normal disassembly would not reveal the second LDX. Since the SPECIAL entry point must be referenced elsewhere at least once, then its address is added to DISASM's internal address table and a label is assigned. Since DISASM uses Apple's built-in disassembler then it too would have missed the second LDX. Thus it would not have output the leading label for SPECIAL. Your assembler, however, will flag the missing label with an error message during assembly. Its proper location shouldn't be too difficult to determine since the label name contains the target address.

Fortunately the use of 'hidden' code and other forms of software trickery don't appear often in most programs. When it does show up its generally there to satisfy a very special need and rightly deserves some personal attention on your part to determine what's going on.

User-Defined Label Name Tables: The DISASM program diskette contains a label name table with over 100 label names for Apple's Monitor Rom routines and page zero useage as well as some of the DOS and I/O functions. Although the table is by no means complet it surves as a starting point for you to further develop and customize to surve your own needs. The label table is also provided in source code format so that you can use your own assembler to extend it.

Table format consists of 8 bytes per label. The first 2 bytes form the address (hibyte first) and the remaining 6 bytes are used for the associated label name (negative ASCII). There is no restriction on name table length. The table is terminated by an FFFF address.

FIGURE 1

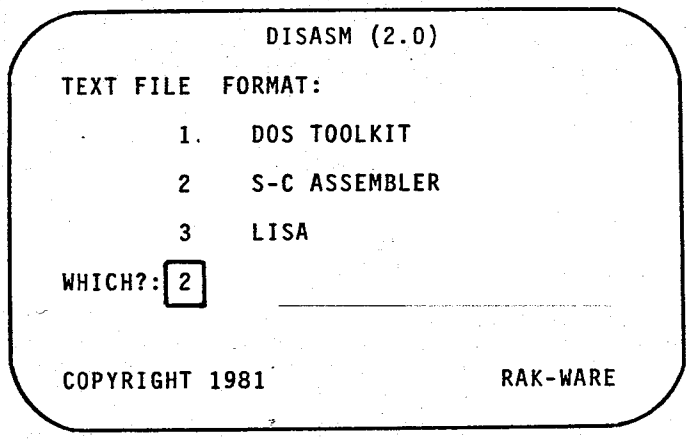
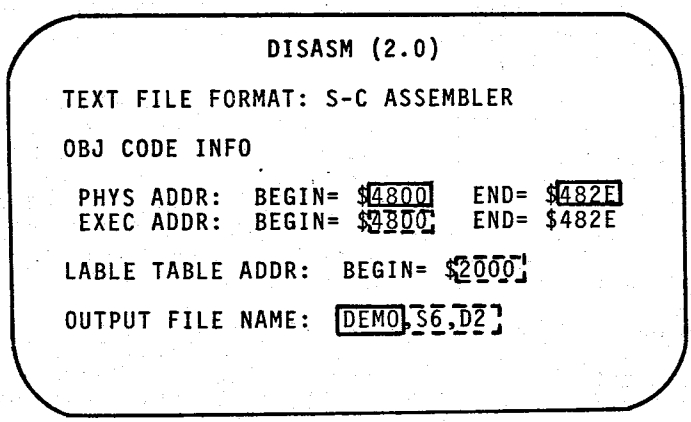


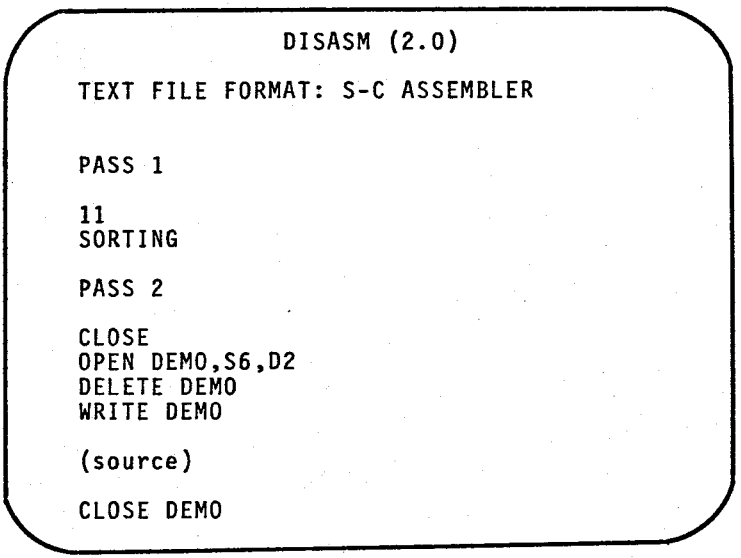
FIGURE 2


DISASM (2.0)
TEXT FILE FORMAT: S-C ASSEMBLER
OBJ CODE INFO
PHYS ADDR: BEGIN= $4800 END= $482E
EXEC ADDR: BEGIN= $4800 END= $482E
LABLE TABLE ADDR: BEGIN= $2000
OUTPUT FILE NAME: DEMO,S6,D2

= Required Entries

= Optional Entries

FIGURE 3



ACKNOWLEDGEMENTS

Apple II, Apple II Plus, Applesoft and DOS Toolkit are products of APPLE COMPUTER CO.

S-C Assembler is a product of S-C SOFTWARE

LISA is a product of PROGRAMMA INTERNATIONAL

USER NOTES